

Real-time Operating Systems and Systems Programming

Networking Lecture 11

Summary

- Recap on Unix IO
- Networking issues

Recap on IO

- Two ways of working with files
 - Unix IO: `open()`, `read()`, `write()`, `close()`
 - System calls to kernel, not buffered, can be interrupted, sometimes won't return everything etc
 - Standard IO: `fopen()`, `fread()`, `fwrite()`, `fclose()`
 - Constructions built on system calls, buffered, widely used, easier

Unix IO

- Why necessary to know
 - Helps to understand other concepts such as process creation anomalies
 - Sometimes necessary to use: file metadata, network programming risks

Unix IO

- File: sequence of bytes
- Open files: `open()`
- Change current file position: `seek()`
- Read and write: `read()`, `write()`
- Close: `close()`

File sharing

- Open files in kernel are in 3 structures:
 - Descriptor Table: unique for a process points to:
 - File Table: shared by processes, holds file position, reference count, points to V-node table:
 - V-node Table: shared by processes, basically holds most of stat() information
- Fork() copies Descriptor Table

What to use

- Use Standard IO calls if you can
- Use Unix IO if you must
 - Meaning: mostly for networking & control/speed

Networking

- TCP/IP protocol
- On hardware level we have network adapter which uses system bus to communicate with memory (usually with DMA)

Internet

- Contains a number of interconnected networks
- Can join LANs and WANs with incompatible technology
- Concerns how a *source host* can send data to *destination host*.
- Solution is a protocol which tells how routers should cooperate to deliver the data
- Naming scheme + Delivery mechanism

Naming Scheme

- Computers are numbered
 - 193.40.252.80
 - Basically a 4-byte number (regular integer)
 - Some numbers have meanings:
 - 127.0.0.1 - localhost
 - 192.168.X.X - LAN address
- DNS service maps names to addresses
 - started in 1988 (before that: hosts.txt)
 - dijkstra.cs.ttu.ee

Getting and creating addresses

- Addresses have structure:

```
struct in_addr {  
    unsigned int s_addr; /* network byte order */  
};
```

- Network byte order: big-endian

```
unsigned long int htonl(unsigned long int hostlong);  
unsigned short int htons(unsigned short int hostshort);
```

```
unsigned long int ntohl(unsigned long int netlong);  
unsigned short int ntohs(unsigned short int netshort);
```

Hostname conversion

```
int inet_aton(const char *cp,  
              struct in_addr *inp);  
char *inet_ntoa(struct in_addr in);
```

Domain names

- Domains are structured
 - dijkstra.cs.ttu.ee -> ee > ttu > cs > dijkstra
- Host entry structures

```
struct hostent {  
    char *h_name; /* official name */  
    char **h_aliases; /* null-terminated array of domains */  
    int  h_addrtype; /* address type AF_INET */  
    int  h_length;    /* address length */  
    char **h_addr_list; /* null terminated array of in_addr structs */  
};
```

- Retrieval and query

```
struct hostent *gethostbyname(const char *name);  
struct hostent *gethostbyaddr(const char *addr, int len, 0);
```

Domain name mappings

- One to one
 - Host has only one name and address
- Multiple domains to one address
 - dragon.ee www.dragon.ee
- Multiple addresses to multiple domains
 - most of google
- Consider when working with host entries

Internet connection

- Communication done by sending streams of bytes over the wire
- Full duplex: you can both read and write
- Point to point: connects a pair of processes
- Socket: endpoint for communication
 - address:port
- Connection is a pair of sockets

Socket interface

- Berkeley sockets
 - developed by their researchers, distributed with Unix 4.2 BSD kernel and distributed to universities and labs
- Socket from the view of kernel: communication endpoint
- Socket from a programs view: an open file

Socket addresses

- Socket address; general and specific

```
struct sockaddr {  
    unsigned short    sa_family;    /* protocol family */  
    char              sa_data;      /* address data */  
}
```

```
struct sockaddr_in {  
    unsigned short    sin_family; /* address family AF_INET */  
    unsigned short    sin_port; /* port number in network byte order */  
    struct in_addr     sin_addr; /* IP address in network byte order */  
    unsigned char     sin_zero[8]; /* pad to sizeof(struct sockaddr) */  
}
```


Overview of interface

- Client

- socket()
- connect()
- read()/write()
- close()

- Server

- socket()
- bind()
- listen()
- accept()
- read()/write()
- close()

socket()

- Creates a socket descriptor

```
// int socket(int domain, int type, int protocol);
```

```
clientfd = socket(AF_INET, SOCK_STREAM, 0);
```

connect()

- Establish a connection with given socket address
- Blocks until successful or error occurs

```
int connect(int sockfd, struct sockaddr *serv_addr, int addrlen);
```

bind()

- Associate a socket with an address and port

```
int bind(int sockfd, struct sockaddr *my_addr, int addrlen);
```

- Convert active socket to listening socket

```
int listen(int sockfd, int backlog);
```

- Accept incoming connection

- note that a new file descriptor is returned; why?

```
int accept(int listenfd, struct sockaddr *addr, int *addrlen);
```

Notes

- When a connection is terminated while it is read, a signal is generated
 - EPIPE: Broken pipe, program terminates unless handled
- There are additional functions to replace read() and write() with sockets
 - send(), recv(): et specify additional flags for sending and receiving data
- For UDP you can use recvfrom() sendto()
 - connect() or bind()/listen() are not needed for them